

Scene Graph APIs: Wired or Tired?

Wes Bethel

R3vis Corporation & NERSC/LBNL

1.0 Abstract

Following widespread adoption and use, the scene graph model has proven to be a popular and powerful development tool because it enables the rapid creation of portable and efficient graphics applications. Unfortunately, not all applications fit within the boundaries imposed by a scene graph model. This panel will examine issues related to scene graph technology.

A class of software tools based upon a scene graph model has assumed an increasingly visible position in the set of resources available for developers. The basic model implemented by scene graph APIs is that a “scene” is built incrementally using an API, then scene-centric operations, such as rendering and picking, are implemented within the underlying infrastructure. From a developer’s point of view, the scene graph model encapsulates and hides numerous complexities of, and specificities of, implementation-dependent resources, such as texture management and the underlying graphics hardware. With such a stratification of resource management, it is possible to achieve both portable and efficient rendering applications.

The purpose of this panel is to examine in a non-partisan, technical fashion, issues related to scene graph technology. During an introductory statement, the scene graph model will be defined in general terms, along with identification of the high-level goals this technology is designed to address. Following, panelists will, in turn, address a more specific set of issues from their respective area(s) of expertise. The scope of these issues range from architectural considerations to applicability for a given class or type of application, as well as alternatives to the scene graph model.

2.0 Panelists

2.1 Andries Van Dam

What is Scene Graph and When is it Appropriate for Use?

A scene graph (aka hierarchical display file) is a retained special-purpose data structure, typically extracted from an application data structure/data base, from which the graphics system can efficiently render/refresh a scene and do first-level interaction handling. Such a “retained mode” is particularly advantageous if the scene doesn’t change considerably between consecutive frames, which is the case, for example, for walk-throughs and fly-overs of largely unchanging scenes, and if certain kinds of performance optimizations are done to drive the underlying immediate mode graphics (hardware) pipeline efficiently. Recent scene graphs have generalized the more traditional ones (e.g., PHIGS) with advances in such areas as viewing models, geometry compression, rendering capabilities, large-model optimization, and multi-processor optimization.

Andy van Dam has been at Brown since 1965, where he co-founded the Computer Science Department and was its first chairman. His interest in scene graphs dates from the late sixties, and he was part of multiple scene graph design efforts, including GPGS (the General Purpose Graphics System) in the early '70s, the SIGGRAPH Core in the mid-'70s, and PHIGS+ in the mid-'80s. He is probably best known for his co-authorship of the reference books *Fundamentals of Interactive Computer Graphics* and *Computer Graphics: Principles and Practice*. He is on the Technical Advisory Board of several companies, including Microsoft Research, and continues to be involved with multiple startups. He co-founded SIGGRAPH and is a Coons Award winner and a member of the National Academy of Engineering.

2.2 Sharon Rose Clay

Using Scene Graphs to Allow Differentiated Hardware to Produce a Differentiated Application

Silicon Graphics has been using scene graph technology for ten years, through three generations of differentiated graphics architectures, and across a diverse product line. Silicon Graphics has used scene graphs to abstract away hardware details and underlying optimizations, allowing applications to get maximum benefit of underlying hardware without being specifically customized to that hardware. While a scene graph may take many shapes and forms, I'll suggest here that the quintessential property is a level of abstraction for the data the structure holds. This level of abstraction can contain semantic information about usage intent, and structural information that can then be used to optimize various operations on that data for a particular type of solution. As our sophistication at Silicon Graphics regarding scene graphs has matured, so has our architectural approach and the problems that we are trying to solve.

Originally, our focus was primarily on enabling software developers and customers to get the most from their differentiated desktop or high-end hardware in a particular application area, such as real-time visual simulation, highly dynamic scene modeling, and image processing.

Building on past success and lessons learned in both scene graph applications and our success in establishing OpenGL as an industry standard Graphics API, we are now focused on developing increasingly sophisticated scene graph architectures with a broader level of standardization than ever before.

This talk will talk about some of the defining lessons we at Silicon Graphics learned in the past in developing scene graph solutions and the problems we are addressing now in our current efforts.

Sharon Rose Clay is Engineering Manager of the Fahrenheit Scene Graph team in the Advanced Graphics Division at Silicon Graphics, Inc. Sharon has been at Silicon Graphics for 10 years, focusing on high performance 3D rendering systems for real-time and interactive applications. She started in the Advanced Graphics microcode group and worked on the development of the VGX, VGXT, and the Reality Engine graphics subsystems and then joined the start of the Silicon Graphics scene graph efforts in 1991. Sharon has been involved with several scene graph products on a variety of graphics architectures for deployment in a variety of applications, including modeling, visual simulation, virtual sets, surgical simulation, urban planning, and location based entertainment. Her hobbies revolve around making real-time a way of life.

2.3 Henry Sowizral

Viewing Scene Graphs As Source Code

Scene graphs work well for scene composition: for placing, orienting, grouping, and procedurally manipulating objects within a scene. This domain-oriented approach makes the application programmer's tasks much simpler.

Unfortunately, this same domain-orientation very frequently results in scene graphs that are poorly structured from a rendering perspective. Such "poorly structured" graphs require the underlying software system to perform significant optimizations. A scene graph's structure can impose subtle ordering constraints. These constraints can preclude possible parallelization or content reordering that could provide better rendering performance. Moreover, because scene graphs tend not to have a strong spatial coherence, spatially-based operations such as picking, collision detection, and culling become rather inefficient.

If we view scene graphs as "source code," then a "compilation" can reorder and optimize the scene description into a far more efficient form, possibly even generating system specific code that exploits the underlying hardware resources such as multiple processors and render pipelines. A compilation can localize and appropriately schedule state-change operations and enable concurrency. By building ancillary data-structures with better spatial coherence, the compilation can also accelerate other operations such as picking and collision detection. The careful semantic design of a scene graph API can simplify the programming task and enable efficient rendering.

Henry Sowizral is a distinguished engineer at Sun Microsystems, Inc., where he is the chief architect of the Java 3D (tm) API. His areas of interest include virtual reality, large model visualization, and distributed and concurrent object-oriented simulation. He has taught numerous tutorials on topics including expert systems and virtual reality at conferences including COMPCON, Supercomputing, VRAIS, and SIGGRAPH. Henry has taught Java 3D at SIGGRAPH, Eurographics, Visualization, JavaOne, and other conferences.

Henry is a co-author of the book *The Java 3D API Specification*, published by Addison-Wesley. He holds a B.S. in Information and Computer Science from the University of California, Irvine, and an M.Phil. and Ph.D. in Computer Science from Yale University.

2.4 Wes Bethel

Title: Scientific Visualization and Scene "Grafting"

Scene graph models and scientific visualization systems can harmoniously coexist, despite a push for optimizations that contribute to dilution of flexibility and extensibility. Taking a step back, we can reference the similarity between traditional dataflow visualization systems and scene graph models. The dataflow model describes a visualization "program"; a scene graph model describes the composition and layout of a collection of geometry and a viewpoint in space. The "root node" of a data flow program is the data "sink," which is usually a renderer. The "root node" of a scene graph is owned by "the system," with "data" pushed out to the leaves of the scene graph. In both, there is a well-defined and (a mostly) deterministic traversal path. Both systems hide complexities from the user, sometimes at the expense of extensibility and flexibility.

As a visualization person, I must be concerned with data modeling and dynamically changing the contents of the scene graph. For example, as a user navigates through data, the subset of data visible to them changes. Using "straight scene graph," since "all the data" is owned by the system, render-time view frustum culling can be used to manage spatial complexity by breaking a big problem into a set of smaller problems. Alternatively, user code can be "grafted" with the scene graph model to provide data based upon the current view. The former method is feasible only when "all the data" fits into memory at render time. The latter method represents a harmonious blending of technology, potentially at the expense of real-time frame rates. Such blending of application and system resources represents one approach to implementing "large model" visualization, the coupling of simulation with visualization, and computational steering. Herein lies the conflict between the scene graph system's inclination to optimize and the developer's need for flexibility.

Scene graph is interesting as a framework for visualization due to its fundamental properties: defined traversal/execution mode and optimized rendering. Visualization clearly benefits from the fastest rendering possible, but at the same time, requires a flexible and extensible framework. Given the disparate needs of a wide range of developers and users, one point of view is that there is no "one true scene graph design" that will satisfy the needs of the graphics and visualization community; there will always be "outliers" that bend and warp the scene graph metaphor beyond a "container for rendering."

Wes Bethel serves the dual roles of Technical Director at R3vis Corporation, a startup dedicated to visualization engineering, and as Staff Scientist at the Lawrence Berkeley National Laboratory/National Energy Research Scientific Computing Center. He has been active in the field of scientific visualization for over 15 years, first in the oil industry in the 1980s, and more recently in the energy research community. He has designed and built graphics, visualization and imaging software for wide range of commodity and custom graphics hardware. Bethel earned his MS degree in Computer Science in 1986 from the University of Tulsa, under Sam Uselton, and has received numerous awards for his contributions to the field of scientific visualization.

2.5 Carl Bass

Every application, worthy of that moniker, has a complex object model, but one that is probably not optimized for graphics display. In my experience, there are several characteristics that determine whether or not scene graph display technology is appropriate for a particular use. Is this a new application or are you retro-

fitting an existing application? Is there enough memory to allow for the duplication of data that is associated with scene graphs? How well does the scene graph accommodate editing operations and data changes common to your application? What hardware functionality is being obscured by the scene graph's attempt to be easy to use? How difficult is it to replace a particular scene graph when better technology enters the market?

Carl Bass is Chief Technical Officer and Vice President of Autodesk. Before joining Autodesk, Carl Bass was a founder of Ithaca Software and co-creator of HOOPS, one of the first commercially available scene graph components, back when they were known as display lists.

2.6 Michael T. Jones

Innovation Above and Beyond the Scene Graph

Scene graphs complement low-level hardware abstraction APIs by providing compound structures and common graphics utilities to application developers. They can add convenience, efficiency, portability, and structure to programming projects and ease the use of differentiated hardware benefits that have created a following among application developers. These virtues, admirable as they are, cannot hide the fact that scene graphs are merely the second rung on a ladder too short to reach the goal of continued innovation.

The graphics toolmaker's quest to fuel advancement is no longer about discovering how to define cameras, traverse spatial data structures, provide concurrency and synchronization, or allow for extensibility and customization. Nor can the path be one of repetition and refinement; restating past successes in new languages or programming paradigms is treading water, not swimming. Viewing the development landscape from the peak of successful scene graph systems shows that there is a further, and much greater, mountain to climb.

The character of this next level of graphics development tools derives from the intrinsic structure of graphical applications and the context of their deployment. Designing tools with this view illuminates inherent flaws in the scene graph approach. Solutions to these flaws exist only at a higher level of abstraction, that of the graphical application platform. This framework approach leverages all that is good in scene graphs while avoiding their weaknesses, adding a third rung to the graphics development ladder to create a powerful development environment that succeeds where scene graphs alone must fail.

Michael T. Jones is the Chief Technology Officer of Intrinsic Graphics (www.intrinsic.com), an innovative Silicon Valley computer graphics company. He directs the company's technological path and its relationships with corporate partners. Prior to Intrinsic Graphics, Michael was Director of Engineering at Silicon Graphics with responsibility for SGI's graphics software OpenGL, PC OpenGL, OpenGL Optimizer, OpenGL Volumizer, Open Inventor, IRIS Performer, ImageVision Library, Molecular Inventor, and Cosmo3D. He also managed SGI's graphics projects with Sun Microsystems (Java3D), the OpenGL ARB (OpenGL++), and Microsoft (Fahrenheit Low-Level, Scene-Graph, and Large Model Visualization APIs). He has a number of patents issued and pending. An autodidact and avid reader, Michael discovered computers and wrote his first assembler program in the fourth grade, fell in love with computer graphics shortly thereafter, and has programmed on a paid professional basis since the seventh grade.

2.7 Brian Hook

Is Scene Graph Technology Good for Games Developers?

Top tier consumer 3D action games compete on many different fronts, including content, performance, gameplay, and striking visual displays. Unfortunately the goals of high performance and truly stunning graphics are often mutually exclusive, and require compromises and techniques that may not be readily apparent to scene graph API authors, or applicable to a wide range of applications. With our particular genre, first person 3D action games, performance needs dictate that we must pay heavy attention to the issues of occlusion, visibility determination, and level of detail; while at the same time the need for leading edge graphics require that we present realistic surface textures, geometry, and lighting, most likely using techniques that have never been implemented.

A general purpose solution to a specific problem can not compete with a specific solution to that specific problem, assuming all else is equal (i.e. talent, hardware, and resources). Applications can make compromises of performance vs. quality vs. generality that are far more fine grained and appropriate than similar decisions made by a general purpose library. While scene graph APIs can allow applications to arbitrarily extend their features into unknown areas, doing so (e.g. via subclassing or callbacks) subverts many of the claimed benefits of a scene graph API. This then begs the question "If I'm spending the majority of my time going through back doors because the API doesn't have the features I need, what is it offering me in the first place?"

Brian Hook is a programmer at Verant Interactive, makers of popular massively multiplayer online role playing games for the consumer market. Prior to working at Verant he was a programmer at id software working on 3D graphics rendering technology for consumer 3D action games. Brian has also been a contractor for Silicon Graphics and a software engineer at 3Dfx Interactive. A former columnist for Game Developer Magazine, he is also a frequent speaker at the Game Developer's Conference. Brian attended the University of Florida.